(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平9－204300

(43)公開日　平成９年(1997)８月５日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	9/06	５３０		Ｇ０６Ｆ	9/06	５３０Ｔ
		４１０				４１０Ｅ
	9/44	５３０			9/44	５３０Ｐ

審査請求　未請求　請求項の数10　ＯＬ　（全 13 頁）

(21)出願番号　特願平8－256244

(22)出願日　平成８年(1996)９月27日

(31)優先権主張番号　０８／５７６７３０

(32)優先日　1995年12月21日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　マーチン・ポール・ナリー
アメリカ合衆国27609、ノースカロライナ州、ラレイ、ステイリー・コート　705

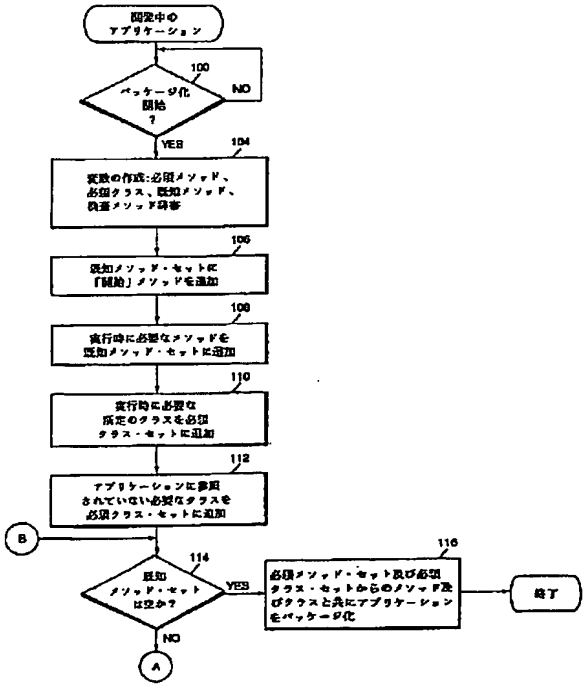
(74)代理人　弁理士　合田　潔　（外２名）

(54)【発明の名称】　オブジェクト指向アプリケーション用パッケージ化アルゴリズム

(57)【要約】

【課題】　アプリケーションのメモリ必要量を低減するために強化されたパッケージ化アルゴリズムを提供する。

【解決手段】　アルゴリズムは、アプリケーションを実行するために必要な開発環境からメソッドとクラスの必要最小限の組を決定する。その後アプリケーションはこれら必要なメソッド及びクラスのみと共にパッケージ化される。得られるアプリケーションは、サイズが縮小されるので、アプリケーションを記憶するために必要なメモリの低減に寄与し、実行時のアプリケーションの性能を向上せる。



【特許請求の範囲】
【請求項１】アプリケーションの実行に必要な開発環境からコードの組を決定するためのコンピュータ読取り可能なコードであって、
前記アプリケーションの実行に必要な既知の１のコンポーネントを識別する第１のサブプロセスと、
前記アプリケーションの実行のための前記既知の１のコンポーネントにより必要とされるコンポーネントを識別する第２のサブプロセスと、
前記第２のサブプロセスにおいて識別された前記コンポーネントにより必要とされるコンポーネントを識別しかつ該識別されたコンポーネントが実行のためのものである第３のサブプロセスとを有するコンピュータ読取り可能なコード。
【請求項２】前記第１のサブプロセスにより、ユーザが少なくとも１つの既知のコンポーネントを識別することができる請求項１に記載のコンピュータ読取り可能なコード。
【請求項３】前記第１のサブプロセスが、前記アプリケーションのコンポーネント自体を識別するために該アプリケーションのコンポーネントを照会し、該識別されたコンポーネントを既知のコンポーネントとして用いる請求項１に記載のコンピュータ読取り可能なコード。
【請求項４】前記アプリケーションが、開発中のオブジェクト指向アプリケーションであり、前記コンピュータ読取り可能なコードがさらに、前記既知のコンポーネント及び前記識別されたコンポーネントのみを含むように前記アプリケーションをパッケージ化する第４のサブプロセスを有する請求項１に記載のコンピュータ読取り可能なコード。
【請求項５】前記識別されたコンポーネントの１つのインスタンスのみが、前記パッケージ化されたアプリケーションに含まれる請求項４に記載のコンピュータ読取り可能なコード。
【請求項６】任意のアプリケーションを実行するために必要な共通のコンポーネントを識別する第５のサブプロセスを有する請求項４に記載のコンピュータ読取り可能なコード。
【請求項７】前記アプリケーションがメソッド及びクラスを有するオブジェクト指向アプリケーションであり、前記既知のコンポーネントがメソッドであり、そして前記第２のサブプロセスがさらに、該既知のメソッドにより参照されるメソッド及びクラスを識別する請求項１に記載のコンピュータ読取り可能なコード。
【請求項８】開発中のオブジェクト指向アプリケーションを実行するために必要な最小限のクラス及びメソッドの組を決定するコンピュータ読取り可能なコードであって、
既知のメソッドを識別しかつ該既知のメソッドを既知メソッド・リストに追加する第１のサブプロセスと、
前記既知メソッド・リストからメソッドを選択しかつ該既知メソッド・リストから該選択されたメソッドを削除する第２のサブプロセスと、
前記選択されたメソッドが必須メソッド・リストに含まれるか否かを判断し、含まれる場合は、前記第２のサブプロセスを繰り返す第３のサブプロセスと、
前記選択されたメソッドを前記必須メソッド・リストに追加しかつ該選択されたメソッドにより参照されるメソッドのリストを作成する第４のサブプロセスと、
参照されるメソッドの各々に関して、各参照されるメソッドの実装クラスが必須クラス・リストに含まれるか否かを判断する第５のサブプロセスと、
前記第５のサブプロセスにおいて前記必須クラス・リストにその実装クラスが含まれないと判断された場合、各参照されるメソッドを前記既知メソッド・リストに追加する第６のサブプロセスと、
前記第５のサブプロセスにおいて前記実装クラスが必須クラスであると判断された場合、前記実装クラスの各々に対応する検査メソッド・リスト内のエントリを検査し、エントリがなければ該検査メソッド・リスト内にエントリを作成し、そして該検査メソッド・リスト内のその実装クラスについてのエントリに実装メソッドを追加する第７のサブプロセスと、
前記選択されたメソッドにより参照されるクラスの第１のクラス・リストを作成する第８のサブプロセスと、
前記第１のクラス・リスト中の参照されるクラスの各々について、該参照されるクラス及びその全てのスーパークラスを含む第２のクラス・リストを作成する第９のサブプロセスと、
前記第２のクラス・リスト中の各クラスに関して、該クラスが前記必須クラス・リストに含まれるか否かを判断し、含まれない場合は、該クラスを該必須クラス・リストへ追加し、前記検査メソッド・リスト中の該クラスに対応するエントリを検査し、そして、エントリが存在するならば、該エントリ中の各メソッドを該既知メソッド・リストへ追加し、該エントリを該検査メソッド・リストから削除する第１０のサブプロセスと、
前記既知メソッド・リストが空となるまで、適宜前記第２乃至第１０のサブプロセスを繰り返す第１１のサブプロセスとを有するコンピュータ読取り可能コード。
【請求項９】演算環境において、作成中のアプリケーションのためにアプリケーション開発環境から必要なコンポーネントの組を決定するシステムであって、
前記アプリケーションの実行に必要な第１のコンポーネントを識別する手段と、
前記アプリケーションに含めるために、ネームにより参照されるか又は前記第１のコンポーネントにより必要とされる第１の組のコンポーネントを識別する手段と、
ネームにより参照されるか又は前記第１の組の個々のコンポーネントにより必要とされる第２の組のコンポーネ

ントを識別しかつ該識別された第２の組の個々のコンポーネントがその後前記アプリケーションに含めるためのものである手段と、
前記第１のコンポーネント、前記第１の組のコンポーネント、及び前記第２の組のコンポーネントから必要なコンポーネントのリストを作成する手段とを有するシステム。
【請求項１０】前記必要なコンポーネントの各々を、前記リスト上に１回だけ含める手段を有する請求項９に記載のシステム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、データ処理システムにおける改良に関し、特に、無関係なクラス又はメソッドを含むことなくオブジェクト指向アプリケーションをパッケージ化(packaging)できるオブジェクト指向アプリケーション開発ソフトウェアのためのパッケージ化アルゴリズムに関する。
【０００２】ソフトウェア・アプリケーションを書いたり開発したりするためにオブジェクト指向言語を用いることは、最近では一般的なこととなりつつある。オブジェクト指向技術には、手続き型言語に比べて幾つかの利点が有しており、比較的使い易いこと、修正が容易であること、アプリケーションからアプリケーションへのコンポーネントの再利用性に優れていること等がある。オブジェクト指向ソフトウェア・アプリケーションは、通常、オブジェクト指向開発環境を用いるソフトウェア開発者により開発される。オブジェクト指向開発環境(例えば、IBM CorporationによるSmalltalkのためのVisual Age)は、通常、Smalltalk又はC++等のオブジェクト指向言語、ブラウザ等の様々な開発ツール、バージョン管理機能、デバッガを含み、さらに、開発者がアプリケーションに求める様々な機能を提供する再利用可能なオブジェクト指向のクラス、コンポーネント(components)、パーツ(parts)、及び／又はフレームワークの組を含む。Smalltalk言語は、完成したアプリケーションをコンピュータ上で実行するために必要な基本的機能を含む仮想マシンとして知られる基本的なエンジン（通常、アプリケーションと共にパッケージ化される）を含み、そして再利用可能なオブジェクト指向のクラス、コンポーネント、パーツ、及びフレームワークの豊富な組を含む。開発者は、所望の機能をもったアプリケーションを作成するために、基本的に、利用可能なクラス、コンポーネント、パーツ、及びフレームワークから所望するクラス(オブジェクト)のインスタンスを一緒に引き出す。
【０００３】オブジェクト指向言語の一態様は、各クラスが、通常、相互依存しており、継承として知られる特性をもつことができるものである。さらに、機能については、通常、オブジェクトのうちメソッドとして知られる部分に与えられる。従って、特定のオブジェクトについてのメソッドは、機能を与える別のオブジェクト又はクラスの中のメソッドに依存する可能性がある。このようなオブジェクト指向言語の特性は、効率を求められるパッケージ化方式に対する可能性を与える。
【０００４】これまで、完成したアプリケーションのパッケージ化は、厳密な技術ではなかった。無難な方法として、アプリケーションは、エンドユーザのコンピュータにより実行可能なそのアプリケーションの実行時バージョンを作成する基本的なエンジンと共に最小の判断をもつ完全標準クラス・ライブラリとパッケージ化される。オブジェクト指向言語の欠点の１つは、アプリケーションが適切に実行されることを確保するために、従来技術のパッケージ化アルゴリズムが余り差別化されておらず、その中に含まれるクラスはその開発環境内の再利用可能なクラス・ライブラリからのものだという点である。従って、パッケージ化されたオブジェクト指向アプリケーションのサイズが比較的大きくなってしまうため、エンド・ユーザのコンピュータ上で大きなメモリが必要となる。さらに、アプリケーションのサイズが比較的大きいために、そのアプリケーションが実行されるとき所望するよりも低い性能しか得られない結果となる。
【０００５】従って、アプリケーション開発ツールに用いられるパッケージ化アルゴリズムにおいて、パッケージ化アプリケーションに含めるためのクラスをその開発環境からさらに精細に選択可能とすることにより、メモリの必要性を低減して性能を強化することが必要とされている。
【０００６】
【発明が解決しようとする課題】以上により、本発明の目的は、アプリケーションに必要なメモリを低減するべく、強化されたパッケージ化アルゴリズムを提供することである。
【０００７】本発明の別の目的は、パッケージ化アルゴリズムを用いてパッケージ化されたアプリケーションの性能を強化するような、強化されたパッケージ化アルゴリズムを提供することである。
【０００８】本発明の更に別の目的は、アプリケーションの実行に必要なメソッド及びクラスのみを含む、アプリケーション開発環境のためのパッケージ化アルゴリズムを提供することである。
【０００９】本発明の更に別の目的は、以下の説明及び図面並びに本発明の実施例により明らかかする。
【００１０】
【課題を解決するための手段】上記の目的を達成するために、本発明は、アプリケーションを実行するために必要な開発環境から一組のコードを決定するための、コンピュータの読取り可能なコードを提供する。このコンピュータの読取り可能なコードは、そのアプリケーションを実行するために必要な既知のコンポーネントを識別する第１のサブプロセスと、そのアプリケーションの実行

のためにその既知のコンポーネントにより必要とされるコンポーネントを識別する第２のサブプロセスと、第２のサブプロセスにおいて識別されたコンポーネントにより必要とされるコンポーネントを識別しかつ該識別されたコンポーネントが実行のためのものである第３のサブプロセスとを有する。

【００１１】第１のサブプロセスによりユーザは、少なくとも１つの既知のコンポーネントを識別することができ、又はアプリケーションのコンポーネントを照会することによりそれ自体を識別することができ、そして識別されたコンポーネントを既知のコンポーネントとして用いる。アプリケーションは、開発中のオブジェクト指向アプリケーションであってもよく、さらに、コンピュータの読取り可能なコードは、既知のコンポーネント及び識別されたコンポーネントのみを含むようにアプリケーションをパッケージ化する第４のサブプロセスを有する。好適には、識別されたコンポーネントのうち１つのインスタンスのみがそのパッケージ化されるアプリケーションに含まれる。さらに、コンピュータの読取り可能なコードは、任意のアプリケーションを実行するために必要な共通コンポーネントを識別する第５のサブプロセスを有する。アプリケーションは、メソッド及びクラスを有するオブジェクト指向アプリケーションとすることができ、既知のコンポーネントはメソッドであり、第２のサブプロセスはさらに、既知のメソッドにより参照されるメソッド及びクラスを識別する。

【００１２】加えて、本発明は、作成されるアプリケーション用のアプリケーション開発環境から必要な一組のコンポーネントを決定するための、コンピュータ環境におけるシステムを提供する。このコンピュータ環境におけるシステムは、アプリケーションを実行するために必要な第１のコンポーネントを識別する手段と、ネームにより参照されるか又は第１のコンポーネントにより必要とされる第１の組のコンポーネントをアプリケーションに含めるべく識別する手段と、ネームにより参照されるか又は第１の組の個々のコンポーネントにより必要とされる第２の組のコンポーネントを識別しかつ該識別された第２の組の個々のコンポーネントはその後アプリケーションに含めるためのものである手段と、第１のコンポーネント、第１の組のコンポーネント及び第２の組のコンポーネントから必要なコンポーネントのリストを作成する手段とを有する。さらに、このシステムは、必要なコンポーネントの各々をリスト上に１回だけ含める手段を有してもよい。

【００１３】

【発明の実施の形態】本発明は、様々なオブジェクト指向開発言語により、例えば、Smalltalk、C++、scriptX等により実行可能である。図１は、本発明を実施可能な代表的ハードウェア環境のブロック図である。図１の環境は、汎用単一ユーザのコンピュータ・ワークステーションであり、例えば、パーソナル・コンピュータとその関連の周辺装置等の代表的なものである。この環境は、マイクロプロセッサ１０と、既知の技術によりマイクロプロセッサ１０とワークステーションの各構成要素との間を接続し通信を可能とするために使用されるバス１２とを含む。ワークステーションは、通常、バス１２を介してマイクロプロセッサ１０を１又は複数のインタフェース装置、例えば、キーボード１６、マウス１８、及び／又は他のインタフェース装置２０へ接続するユーザ・インタフェース・アダプタ１４を含む。他のインタフェース装置２０としては、例えば、接触感知画面やデジタイザ・ペン入力パッド等の任意のユーザ・インタフェース装置がある。さらに、バス１２は、表示アダプタ２４を介してＬＣＤ画面やモニタ等の表示装置２２をマイクロプロセッサ１０へ接続する。バス１２は、さらに、ＲＯＭ、ＲＡＭ等を含むメモリ２６へマイクロプロセッサ１０を接続する。

【００１４】本発明で用いるソフトウェア・プログラミング・コードは、一般的にはスタンドアロン型ワークステーション環境のメモリ２６に記憶される。クライアント／サーバ環境においては、ソフトウェア・プログラミング・コードを、サーバに関連するメモリに記憶することができる。ソフトウェア・プログラミング・コードは、データ処理システムと共に用いるディスケットやＣＤ－ＲＯＭ等任意の様々な既知の媒体上で具現化することができる。ソフトウェア・プログラミング・コードは、このような媒体上に分散させることができ、あるいは、所与の形式のネットワークを介して１つのコンピュータ・システムのメモリから他のコンピュータ・システムへ分配され、それを他のシステムのユーザが利用することができる。ソフトウェア・コードを物理的媒体上に具現化したりネットワークを介してソフトウェア・コードを分配するこのような技術及び方法は、周知であり、ここでは詳細に説明しない。

【００１５】以下、Smalltalk開発環境部分である好適例において本発明を説明する。Smalltalk開発環境は、オペレーティング・システム及びハードウェアの任意の様々な組合せに対して動作可能であるので、特定のオペレーティング・システム及びハードウェアと関係なく説明することとする。Smalltalkは、動的オブジェクト指向言語であり、かつ純粋なオブジェクト指向言語といわれている。なぜなら、Smalltalkは、継承、ポリモルフィズム等のオブジェクト指向言語の基本的定義の全てを具備するからである。これらの概念については、ここで説明しない。他の幾つかのオブジェクト指向言語、例えば、Ｃ＋＋等は、アーキテクチャにおいてやや異なっている。しかしながら、他のほとんどのオブジェクト指向言語においても前述のパッケージ化の問題が同様に存在しており、本明細書に開示される解決手段は、これらの他のオブジェクト指向言語に対しても直接的に適用可能

である。

【００１６】図２は、IBM社によるVisualAge等のSmalltalkオブジェクト指向開発環境の基本アーキテクチャを示す図である。オブジェクト指向開発環境４０自体がアプリケーションであり、基礎的オペレーティング・システム４２上で実行される。開発環境４０の中で仮想マシンとして知られる部分４４が、オペレーティング・システムと相互的に動作する。Smalltalk開発環境は階層的であり、開発環境４０のイメージ部分４６には、仮想マシン４４へ結合された階層的クラスが続き、仮想マシン４４の上で論理的に実行されているものと見ることができる。イメージ４６は開発環境４０の一部であり、開発者はオブジェクト指向アプリケーションを開発するためにイメージ４６と相互的に作業を行う。開発環境４０のイメージ部分４６は、様々な階層に設けられる多様なクラスを含み、これらは多くの異なるレベルにおける機能を提供する。高レベルにおいては、クラス全体の組を、開発者の所望する極めて完成された機能を提供するフレームワークから構成することができ、開発者は、それを開発中のアプリケーションへ引き入れることができる。一方、その機能は、開発環境によるフレームワークのような整然としたパッケージの形で設けることはできない。そのため開発者は、開発中のアプリケーションのために、比較的低レベルのクラスや個々のクラスを組み合わせたり、所望の機能を作るために新たなクラスを書いたりする必要がある。

【００１７】イメージ４６は、アプリケーション開発ツールも含み、それは、異なる環境においては異なっている。これらのツールは、クラス及びメソッドを見るためのクラス・ブラウザ、累積的開発を可能としかつ開発中のアプリケーションを保存するバージョン制御システム、開発環境４０を用いて作成されたアプリケーションをデバッグするためのデバッガ等を含む。さらに、開発環境は、Smalltalkコンパイラを含み、これは、アプリケーションの各部分をリンクしたりコンパイルしたりする。Smalltalkはインタプリタ言語であるので、アプリケーションの各部分はバイト・コードの形態のままであり、これらのコードは実行中に実行時エンジンにより翻訳される。

【００１８】図３は、オブジェクト５０の工業規格表示である。オブジェクト５０のメソッド５２は機能を提供し、一方、データ部分５４はオブジェクト５０に関するデータを含む。オブジェクトは、多くのクラスからなる階層から得られる１つのクラスのインスタンス化であり、開発者がアプリケーションにおいて用いるために指定したものである。１つのアプリケーションにおいて、同じクラスを何回でも用いることができる。

【００１９】図４は、多数のクラス６２からなる階層６０を示す図である。オブジェクト指向階層は、スーパークラス及びサブクラスの概念を用いる。１つのクラスは、クラス階層における上層のクラス（スーパークラス）から全ての変数とメソッドを継承する。継承するクラスは、そのスーパークラスのサブクラスと称される。

【００２０】このアーキテクチャの見地からすると、従来技術によるパッケージ化アルゴリズムは、非常に保守的にパッケージ化を処理してきた。従来のアルゴリズムは、アプリケーションをパッケージ化する際にいずれのクラスを含んでいるのかに関して、それほど差別化しない傾向がある。この手法の場合、必要なクラスを落とすことは滅多にないが、この手法が安全性の面に関して誤って強く働く結果、アプリケーションが扱いにくい大きさとなって多くのメモリを必要とすることとなる。また、時には、最適な性能が低下してしまうことになる。

【００２１】従来技術による場合のように、開発者が、開発環境においてアプリケーションの開発を終了したとき、開発者はその開発環境内でパッケージ化機能を具現化する。パッケージ化アルゴリズムは、実行可能なアプリケーション中に、実行時エンジンと共にいずれのクラスを含めるべきかを決定する。本発明のパッケージ化アルゴリズムについて、図５乃至図８の流れ図を参照して説明する。

【００２２】本発明のパッケージ化アルゴリズムは、アプリケーション及びその関連する再利用可能なコード・ライブラリ内のコードを解析することにより、そのアプリケーションを実行可能とするために必要なコードの最小のサブセットを決定する。アルゴリズムは、ステップのリンク及びパッケージ化に用いられ、そしてアプリケーションを実行するために必要なクラス及び／又はメソッドの最小限定セットを決定する。以下の説明においては、最小の限定されたクラス及びメソッドを、「必須クラス」及び「必須メソッド」と称する。本発明では、クラス（オブジェクト）のインスタンスが互いにメッセージを送り合うような方法でオブジェクト指向アプリケーションが実行されるという事実を利用する。従って、多くの必須クラス及び必須メソッドが、相互に関連している。パッケージ化されたアプリケーションは、仮想マシン、イメージ及びクラス・ライブラリに亘る開発環境からの必須クラス及び必須メソッドを含み、そして実行時にアプリケーションを実行するために必要な仮想マシンからのコードを含むことになる。実行に必要な仮想マシンからのコードは、通常、全てのSmalltalkアプリケーションについて不変である。

【００２３】パッケージ化されるアプリケーションのイメージ部分からの最初の「開始メソッド」は、インプリメンテーション及び言語に従って開発者が規定する必要がある。開始メソッドから、アルゴリズムは、そのアプリケーションにおけるクラス及び他のメソッド並びに再利用可能なクラス・ライブラリの全てを参照することにより、必須メソッド及び必須クラスの組を決定する。

【００２４】そして、開発者がアプリケーションの開発

を終了するとき、その開発者は、開発環境に対してパッケージ化を開始するようにコマンドを入力する（ステップ１００）。その後、プログラムは、多数の変数を作成する。これらの変数には、必須メソッド・セット、必須クラス・セット、既知メソッド・セット、及び検査メソッド辞書が含まれる。これら全ての変数は、初期においては空である（ステップ１０４）。開発環境からのメソッド及びクラスがそのアプリケーションの実行に必要であるとアルゴリズムが判断すると、それらのメソッド及びクラスが、それぞれ必須メソッド・セット及び必須クラス・セットに追加される。アプリケーションは、後に、これらのセットに規定されたメソッド及びクラスのみとパッケージ化されることになる。既知メソッド・セットに入れられたメソッドは、（既に含まれていない場合は）必須メソッド・セットに追加された上、更に必要なクラス及び他のメソッドを識別するためにアルゴリズムに従って処理されることになる。かつ／又は、既知メソッド・セット内のメソッドは、他の必要なクラス及びメソッドを識別するためにそれ自体で処理される。検査メソッド辞書は、必要なメソッドの識別を支援するために用いられる。後述するように、他の変数もまたアルゴリズムにより用いられる。

【００２５】ユーザにより又は自動的照会においてプログラム自体により識別された１又は複数の開始メソッドは、既知メソッド・セットに追加される（ステップ１０６）。前述のように、Smalltalkにおいては、仮想マシン及びイメージの既知の部分は、いずれかのアプリケーションのアプリケーション・コードを実行するために必要である。一般的に、実行時アプリケーションを実行するために必要なメソッド及びクラスは、既知である。従って、必須実行時メソッドは、既知メソッド・セットに追加され（ステップ１０８）、実行時実行のための必須クラスは必須クラス・セットに追加される（ステップ１１０）。

【００２６】所与の事例においては、アプリケーション・コードにより必要とされる幾つかのクラスが、アプリケーション開発環境自体に存在しておらず、メソッド・ネームによっては識別できない。例えば、そのようなクラスが、永続的記憶機構内に存在していることがあり、アプリケーションがその実行中にそのクラスを読取る場合がある。しかしながら、永続的記憶機構内のメソッドに対する特定の参照手段がなければ、アルゴリズムはそのようなメソッドやクラスを探索しないことになり、よってパッケージ化のためにそれらを識別しないことになる。従って、この時点で、このような必要なクラスの全てが、必須クラス・セットへ追加される（ステップ１１２）。最もよくあることは、これらのクラスの場所が、アプリケーション開発者により与えられることである。しかしながら、パッケージ化ステップに自動的照会を組み込むことができるので、これにより、このようなクラスの場所を見出すことができ、よってそれらのクラスが自動的に必須クラス・セットへ追加される。

【００２７】アルゴリズムは、既知メソッド・セットの内容を用いて、クラス及びメソッドの最小限定セットの探索を開始する。ステップ１１４において、最終的に既知メソッド・セットが空であると判断されると、クラス及びメソッドの最小限定セットが規定されたことになり、必須メソッド・セット及び必須クラス・セットからのメソッド及びクラスのみを含むようにアプリケーションをパッケージ化することができる（ステップ１１６）。しかしながら、最初は、ステップ１０６においてユーザにより又は自動的に既知メソッド・セットに対して開始メソッド与えられているので、既知メソッド・セットは空ではなく、処理はステップ１１８まで進められる。ステップ１１８において、１つのメソッドが既知メソッド・セット内のメソッドから選択され、既知メソッド・セット内から削除される。このメソッドは、既知メソッド・セット内から無作為に選択してもよく、又は、既知メソッド・セット内に挙げられた最初のメソッドでもよい。その後、必須メソッド・セットが、その選択されたメソッドを既に含んでいるか否かを調べるために検査される（ステップ１２０）。もし含んでいれば、処理はステップ１１４へ戻る。ステップ１２０において、選択されたメソッドが必須メソッド・セットに含まれていないと判断されたならば、処理はステップ１２２へ進み、選択されたメソッドが必須メソッド・セットへ追加される。次に、選択されたメソッドにより呼び出し可能な全てのメソッドのリストが作成される（ステップ１２４）。Smalltalkの特性として、通常のメソッドは、実行中に処理を行うために１又は複数の他のメソッドへメッセージを送る。ステップ１２４において作成されたリストは、これらの他のメソッドから構成される。既知メソッド・セットから選択されたメソッドが他のいずれのメソッドも呼び出さない場合、又はリスト全体のメソッドが処理された後は、ステップ１２６においてリストが空であると判断され、処理はステップ１４０へ跳ぶ。一方、ステップ１２６においてリストが実際に空であると判断されるまでは、リスト内の第１のメソッド又は次のメソッドが参照メソッド変数に割り当てられ、リストから除去される（ステップ１２８）。ステップ１３０において、必須クラス・セットが、参照メソッド変数に含まれるメソッドを具現化するクラスを既に含むか否かが判断される。もし、含んでいれば、参照されるメソッドが既知メソッド・セットへ追加され、参照メソッド変数から除かれる（ステップ１３８）。その後、処理は、リスト内の次のメソッドがあれば、ステップ１２６へ戻る。

【００２８】ステップ１３０において、必須クラス・セットが、参照されるメソッドを具現化するクラスを含んでいなければ、検査メソッド辞書の中にその具現化するクラスに対応するエントリが存在するか否かが判断され

る（ステップ１３２）。ステップ１３２において、そのようなエントリが存在しないと判断されたならば、その具現化するクラスのために検査メソッド辞書に空のセットが追加される（ステップ１３４）。ステップ１３２において、具現化するクラスに対するエントリが存在すると判断されたならば、又は、ステップ１３４においてその具現化するクラスに対応するエントリが作成された後には、参照されるメソッドが検査メソッド辞書内のその具現化するクラスに対するエントリに追加される（ステップ１３６）。参照されるメソッドは変数から除かれ、そして、リスト内に残っているメソッドがあれば処理はステップ１２６へ戻る。

【００２９】ステップ１２６において、リストが空であると判断されたとき、処理はステップ１４０へ進み、（ステップ１１８において）選択されたメソッドにより参照される全てのクラスのリストが作成される。Smalltalkにおいては、他のメソッドへの参照（ステップ１２４～１３８で処理される）は別個になっているので、１つのメソッドが多数のメソッドを参照することができる。以下の記述は、必須クラス・セットに未だ追加されていない、参照されるクラス及びそれらの親クラスすなわちスーパークラスを識別する処理である。

【００３０】先ず、ステップ１４２において、選択されたメソッドがいずれかのクラスを参照するか否か（又は、リストが処理し尽くされたか否か）が判断される。もしそうであれば、処理はステップ１１４へ戻り、既知メソッド・セット中の次のメソッドを処理する。そのリストが少なくとも１つのクラスに対する参照を含んでいれば、リスト中のクラスの１つが参照クラス変数へ割り当てられ、そのリストから削除される（ステップ１４４）。次に、参照クラス変数が、その参照クラス及びその全てのスーパークラスを含むように更新される。所与のクラスに対するスーパークラスを決定する技術は、周知であり本明細書での説明を要しない。

【００３１】ステップ１４８～１６０では、上記のようにして参照クラス変数に挙げられたクラスを処理する。ステップ１５０～１６０の処理によりそのリストが尽くされた後、ステップ１４８により処理はステップ１４２へ戻る。参照クラス変数内のリストにまだクラスが含まれる間は、処理はステップ１５０へ進み、参照クラス変数内の最初の若しくは次の要素すなわちクラスが参照クラス変数ａへ割り当てられ、参照クラス変数から削除される（ステップ１５０）。ステップ１５２において、参照クラス変数ａ内のクラスが既に必須クラス・セットに含まれているか否かが判断される。既に含まれている場合、参照クラス変数内に次のクラスがあればその処理のためにステップ１４８へ戻る。ステップ１５２において、そのクラスが必須クラス・セットにまだ含まれていない場合、そのクラスが必須クラス・セットに追加される（ステップ１５４）。ステップ１５６において、そのクラスに対するエントリが、検査メソッド辞書中の参照クラス変数ａ内に存在するか否かを判断する。このようなエントリが存在しなければ、処理はステップ１４８へ戻る。ステップ１５６においてこのようなエントリが検査メソッド辞書中に存在する場合、処理はステップ１５８へ進み、検査メソッド辞書中の対応するエントリ内の全てのメソッドが、既知メソッド・セットへ追加される。その後、参照クラス変数ａ内のクラスに対応するエントリ及びそのメソッドが、検査メソッド辞書から除かれ、そしてそのクラスが、参照クラス変数ａから除かれる（ステップ１６０）。その後、処理は、参照クラス変数内の残りのクラスのためにステップ１４８へ戻る。

【００３２】ステップ１４８において、参照クラス変数内にそれ以上クラスが挙げられていないとき、処理は、選択されたメソッドにより参照されたクラスのリスト内の残りのクラスのために、ステップ１４２へ戻る。ステップ１４２において、リストが空であると判断されたとき、処理は、既知メソッド・セット内の次のメソッドに関する処理のためにステップ１１４へ戻る。処理がステップ１１４へ戻ったとき、既知メソッド・セット内にそれ以上メソッドが存在しなければ、アルゴリズムは、クラス及びメソッドの必要最小限のセットを識別したことになる。これらのクラス及びメソッドは、それぞれ、必須クラス・セット及び必須メソッド・セットにリストされる。その後、ステップ１１６において、必須クラス・セット及び必須メソッド・セットに識別されたメソッド及びクラスのみがパッケージ化アプリケーションに含まれるように、既知の技術に従ってアプリケーションのパッケージ化が実行される。アプリケーションのパッケージ化のための特定の処理技術については、周知でありかつ現在実用化されているので本明細書では説明しない。

【００３３】本アルゴリズムは、Smalltalkについて特別に提示されたものであるが、プロシージャ又はルーチンがネームにより呼び出される他の言語のパッケージ化ステップに対しても一般的に適用可能である。このアルゴリズムは、C++、ScriptX等の他のオブジェクト指向言語に対しても明らかに適用可能であるが、Cコード・ライブラリ等のパッケージ化にも適用可能である。

【００３４】本発明の好適例を説明したが、当業者によれば、本発明の進歩性概念を理解したならばこの好適例の更なる変形及び修正が行われるであろう。従って、特許請求の範囲には、好適例と共にそのような変形及び修正も含まれる。

【００３５】まとめとして、本発明の構成に関して以下の事項を開示する。

【００３６】（１）アプリケーションの実行に必要な開発環境からコードの組を決定するためのコンピュータ読取り可能なコードであって、前記アプリケーションの実行に必要な既知の１のコンポーネントを識別する第１のサブプロセスと、前記アプリケーションの実行のための

前記既知の１のコンポーネントにより必要とされるコンポーネントを識別する第２のサブプロセスと、前記第２のサブプロセスにおいて識別された前記コンポーネントにより必要とされるコンポーネントを識別しかつ該識別されたコンポーネントが実行のためのものである第３のサブプロセスとを有するコンピュータ読取り可能なコード。

（２）前記第１のサブプロセスにより、ユーザが少なくとも１つの既知のコンポーネントを識別することができる（１）に記載のコンピュータ読取り可能なコード。

（３）前記第１のサブプロセスが、前記アプリケーションのコンポーネント自体を識別するために該アプリケーションのコンポーネントを照会し、該識別されたコンポーネントを既知のコンポーネントとして用いる（１）に記載のコンピュータ読取り可能なコード。

（４）前記アプリケーションが、開発中のオブジェクト指向アプリケーションであり、前記コンピュータ読取り可能なコードがさらに、前記既知のコンポーネント及び前記識別されたコンポーネントのみを含むように前記アプリケーションをパッケージ化する第４のサブプロセスを有する（１）に記載のコンピュータ読取り可能なコード。

（５）前記識別されたコンポーネントの１つのインスタンスのみが、前記パッケージ化されたアプリケーションに含まれる（４）に記載のコンピュータ読取り可能なコード。

（６）任意のアプリケーションを実行するために必要な共通のコンポーネントを識別する第５のサブプロセスを有する（４）に記載のコンピュータ読取り可能なコード。

（７）前記アプリケーションがメソッド及びクラスを有するオブジェクト指向アプリケーションであり、前記既知のコンポーネントがメソッドであり、そして前記第２のサブプロセスがさらに、該既知のメソッドにより参照されるメソッド及びクラスを識別する（１）に記載のコンピュータ読取り可能なコード。

（８）開発中のオブジェクト指向アプリケーションを実行するために必要な最小限のクラス及びメソッドの組を決定するコンピュータ読取り可能なコードであって、

既知のメソッドを識別しかつ該既知のメソッドを既知メソッド・リストに追加する第１のサブプロセスと、前記既知メソッド・リストからメソッドを選択しかつ該既知メソッド・リストから該選択されたメソッドを削除する第２のサブプロセスと、前記選択されたメソッドが必須メソッド・リストに含まれるか否かを判断し、含まれる場合は、前記第２のサブプロセスを繰り返す第３のサブプロセスと、前記選択されたメソッドを前記必須メソッド・リストに追加しかつ該選択されたメソッドにより参照されるメソッドのリストを作成する第４のサブプロセスと、参照されるメソッドの各々に関して、各参照されるメソッドの実装クラスが必須クラス・リストに含まれるか否かを判断する第５のサブプロセスと、前記第５のサブプロセスにおいて前記必須クラス・リストにその実装クラスが含まれないと判断された場合、各参照されるメソッドを前記既知メソッド・リストに追加する第６のサブプロセスと、前記第５のサブプロセスにおいて前記実装クラスが必須クラスであると判断された場合、前記実装クラスの各々に対応する検査メソッド・リスト内のエントリを検査し、エントリがなければ該検査メソッド・リスト内にエントリを作成し、そして該検査メソッド・リスト内のその実装クラスについてのエントリに実装メソッドを追加する第７のサブプロセスと、前記選択されたメソッドにより参照されるクラスの第１のクラス・リストを作成する第８のサブプロセスと、前記第１のクラス・リスト中の参照されるクラスの各々について、該参照されるクラス及びその全てのスーパークラスを含む第２のクラス・リストを作成する第９のサブプロセスと、前記第２のクラス・リスト中の各クラスに関して、該クラスが前記必須クラス・リストに含まれるか否かを判断し、含まれない場合は、該クラスを該必須クラス・リストへ追加し、前記検査メソッド・リスト中の該クラスに対応するエントリを検査し、そして、エントリが存在するならば、該エントリ中の各メソッドを該既知メソッド・リストへ追加し、該エントリを該検査メソッド・リストから削除する第１０のサブプロセスと、前記既知メソッド・リストが空となるまで、適宜前記第２乃至第１０のサブプロセスを繰り返す第１１のサブプロセスとを有するコンピュータ読取り可能コード。

（９）演算環境において、作成中のアプリケーションのためにアプリケーション開発環境から必要なコンポーネントの組を決定するシステムであって、前記アプリケーションの実行に必要な第１のコンポーネントを識別する手段と、前記アプリケーションに含めるために、ネームにより参照されるか又は前記第１のコンポーネントにより必要とされる第１の組のコンポーネントを識別する手段と、ネームにより参照されるか又は前記第１の組の個々のコンポーネントにより必要とされる第２の組のコンポーネントを識別しかつ該識別された第２の組の個々のコンポーネントがその後前記アプリケーションに含めるためのものである手段と、前記第１のコンポーネント、前記第１の組のコンポーネント、及び前記第２の組のコンポーネントから必要なコンポーネントのリストを作成する手段とを有するシステム。

（１０）前記必要なコンポーネントの各々を、前記リスト上に１回だけ含める手段を有する（９）に記載のシステム。

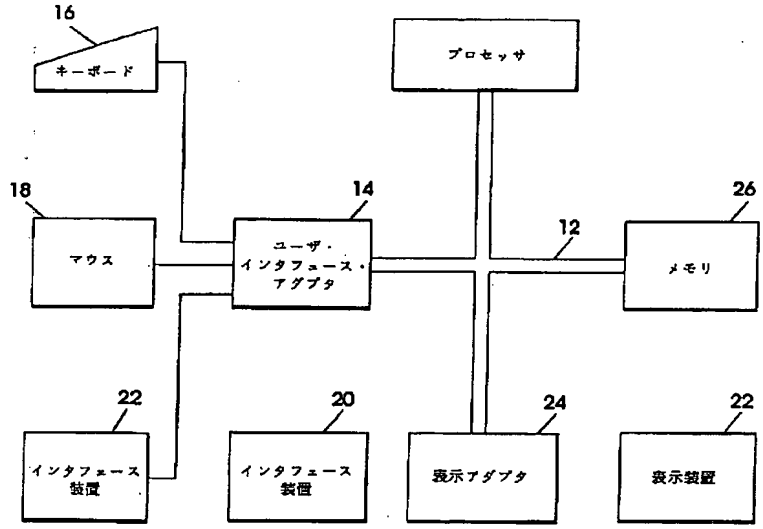
【図面の簡単な説明】

【図１】本発明を実施可能な代表的ハードウェア環境のブロック図である。

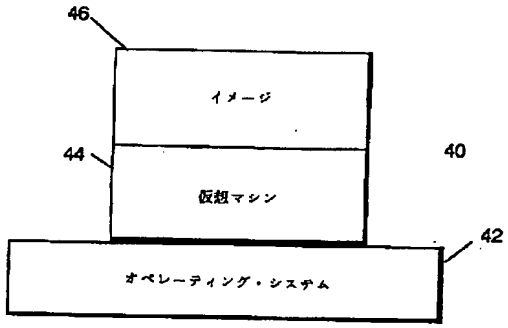
【図２】オブジェクト指向アプリケーション開発環境の

アーキテクチャの図である。
【図３】オブジェクト指向オブジェクトを示す図である。
【図４】オブジェクト指向言語の関連するクラス間における一般的な階層関係を示す図である。
【図５】本発明のパッケージ化アルゴリズムの論理ステップを示す流れ図である。
【図６】本発明のパッケージ化アルゴリズムの論理ステップを示す流れ図である。
【図７】本発明のパッケージ化アルゴリズムの論理ステップを示す流れ図である。
【図８】本発明のパッケージ化アルゴリズムの論理ステップを示す流れ図である。
【符号の説明】
１２　バス
１４　ユーザ・インタフェース・アダプタ
１６　キーボード
１８　マウス
２２　インタフェース装置
２４　表示アダプタ
２６　メモリ
４０　アプリケーション開発環境
４２　オペレーティング・システム
４４　仮想マシン
４６　イメージ
５０　オブジェクト
５２　メソッド
５４　データ
６０　階層
６２　クラス

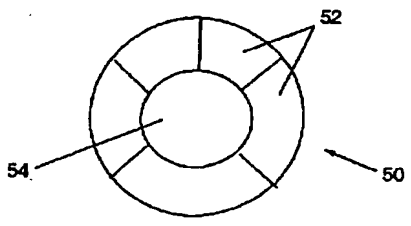
【図１】



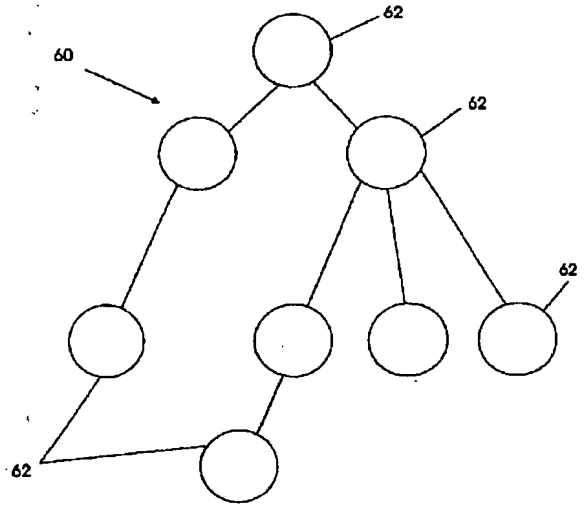
【図２】



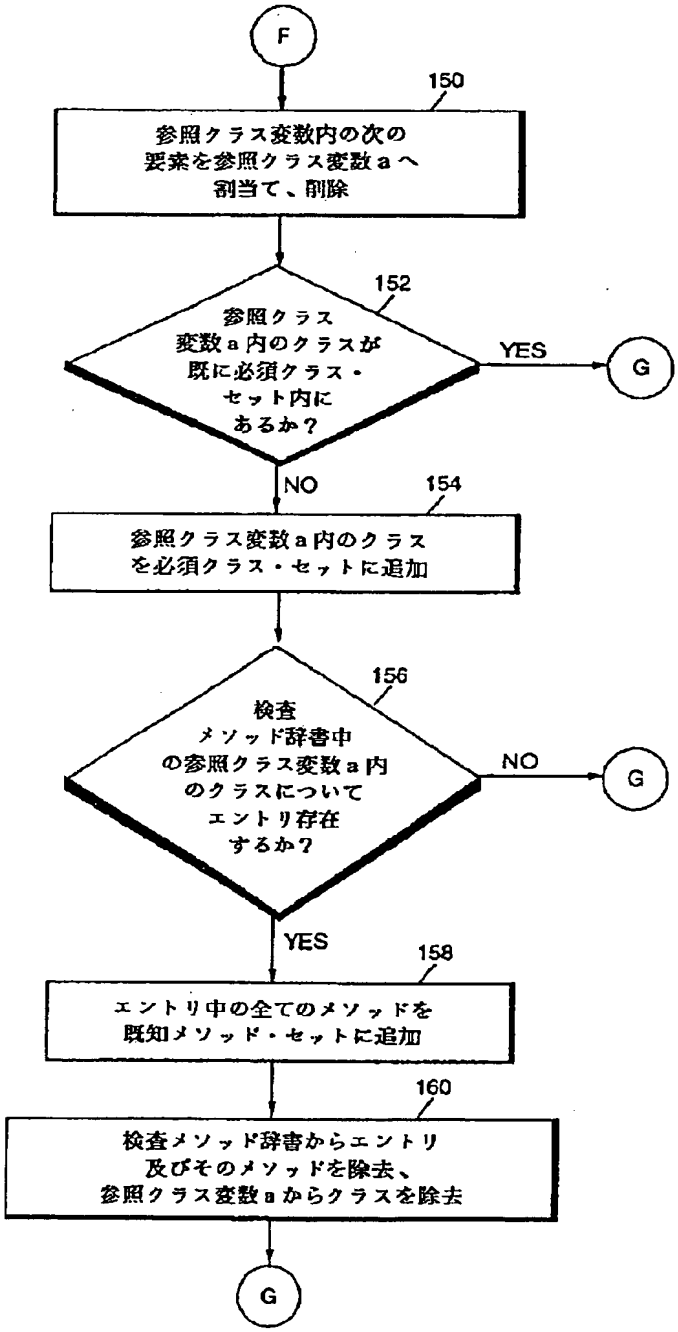
【図３】



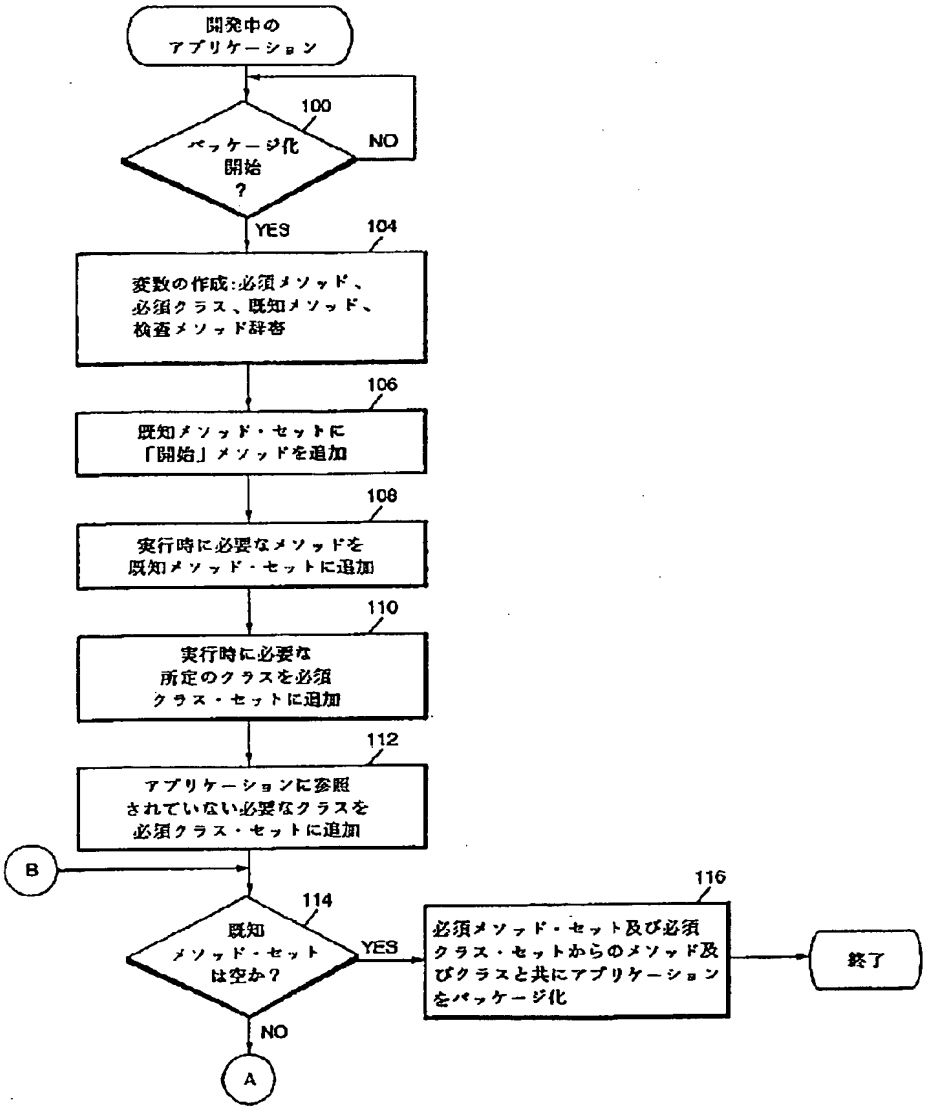
【図4】


60
62
62
62
62

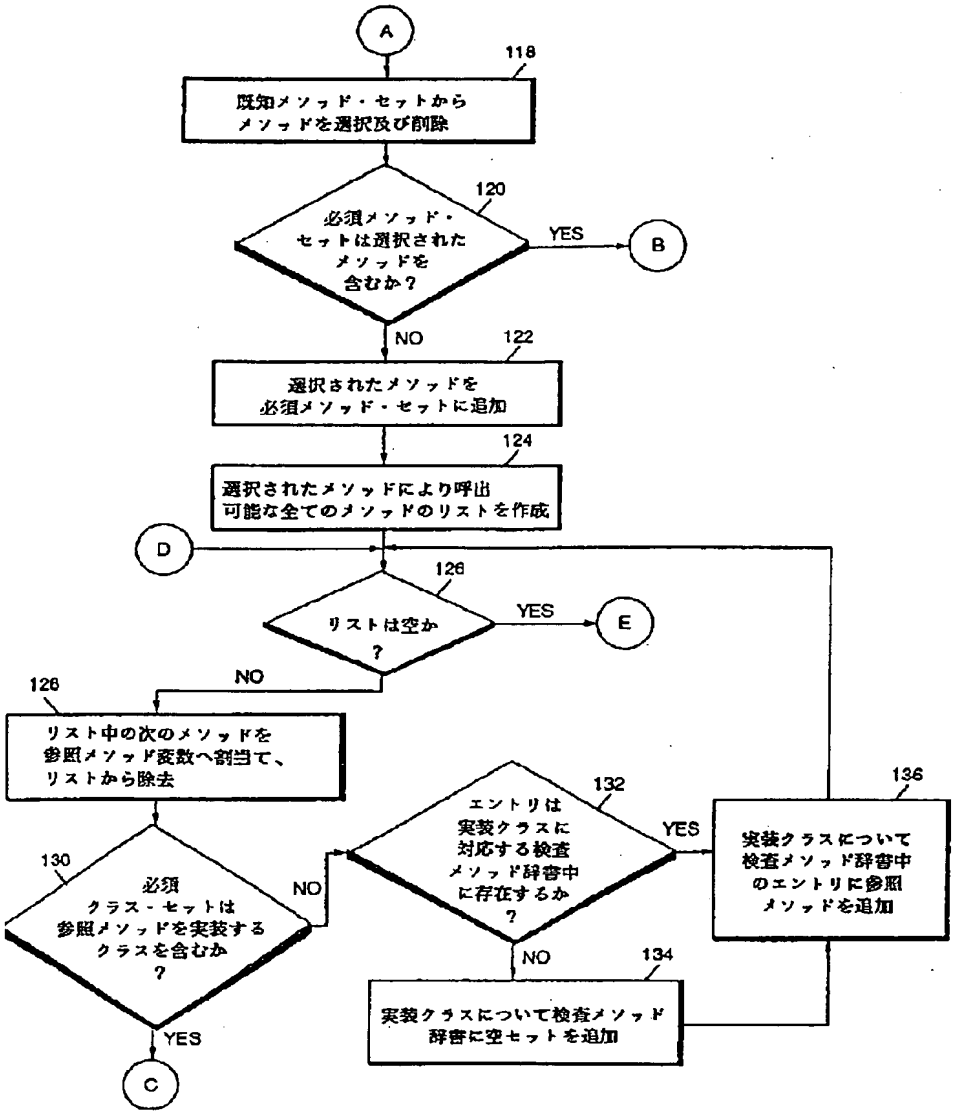
【図8】


F
150
参照クラス変数内の次の要素を参照クラス変数aへ割当て、削除
152
参照クラス変数a内のクラスが既に必須クラス・セット内にあるか？
YES
G
NO
154
参照クラス変数a内のクラスを必須クラス・セットに追加
156
検査メソッド辞書中の参照クラス変数a内のクラスについてエントリ存在するか？
NO
G
YES
158
エントリ中の全てのメソッドを既知メソッド・セットに追加
160
検査メソッド辞書からエントリ及びそのメソッドを除去、参照クラス変数aからクラスを除去
G

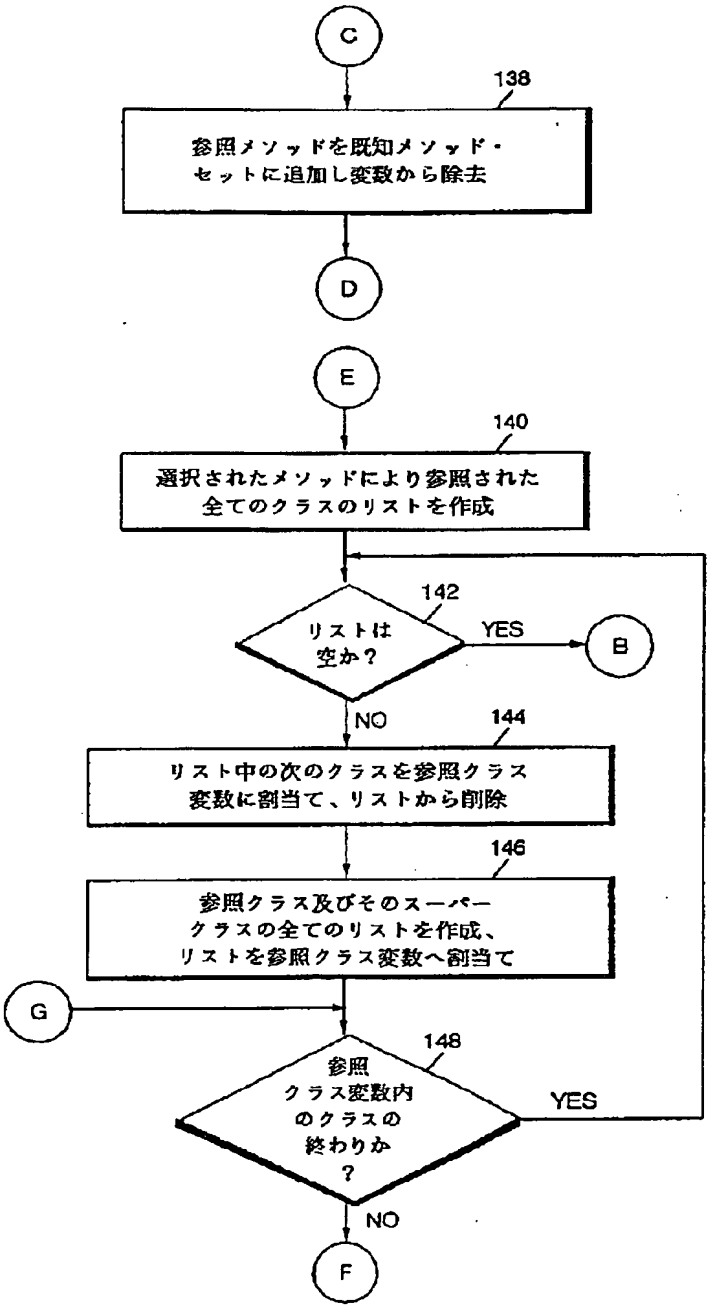
【図５】


開発中の
アプリケーション
100
パッケージ化
開始
？
NO
YES
104
変数の作成:必須メソッド、
必須クラス、既知メソッド、
検査メソッド辞書
106
既知メソッド・セットに
「開始」メソッドを追加
108
実行時に必要なメソッドを
既知メソッド・セットに追加
110
実行時に必要な
所定のクラスを必須
クラス・セットに追加
112
アプリケーションに参照
されていない必要なクラスを
必須クラス・セットに追加
B
114
既知
メソッド・セット
は空か？
YES
116
必須メソッド・セット及び必須
クラス・セットからのメソッド及
びクラスと共にアプリケーション
をパッケージ化
終了
NO
A

【図６】


A
118
既知メソッド・セットから
メソッドを選択及び削除
120
必須メソッド・
セットは選択された
メソッドを
含むか？
YES
B
NO
122
選択されたメソッドを
必須メソッド・セットに追加
124
選択されたメソッドにより呼出
可能な全てのメソッドのリストを作成
D
126
リストは空か
？
YES
E
NO
126
リスト中の次のメソッドを
参照メソッド変数へ割当て、
リストから除去
130
必須
クラス・セットは
参照メソッドを実装する
クラスを含むか
？
NO
132
エントリは
実装クラスに
対応する検査
メソッド辞書中
に存在するか
？
YES
136
実装クラスについて
検査メソッド辞書中
のエントリに参照
メソッドを追加
NO
134
実装クラスについて検査メソッド
辞書に空セットを追加
YES
C

【図７】


C
138
参照メソッドを既知メソッド・セットに追加し変数から除去
D
E
140
選択されたメソッドにより参照された全てのクラスのリストを作成
142
リストは空か？
YES
B
NO
144
リスト中の次のクラスを参照クラス変数に割当て、リストから削除
146
参照クラス及びそのスーパークラスの全てのリストを作成、リストを参照クラス変数へ割当て
G
148
参照クラス変数内のクラスの終わりか？
YES
NO
F

Abstract

The present invention provides intensified packaging algorithm for saving memory. Algorithm determines least pairs of method and class in the development environment for practicing an application. Then the application gets packaged with the method and class in need. With its size curtailed, the resultant application contributes to saving memory, improving and speeding up the performance of the application in practice.